Designing the fastest ACID Key-Value Store

Ashot Vardanian

++ High Load Armenia

Co-organizer

Yandex

Nice to meet you!

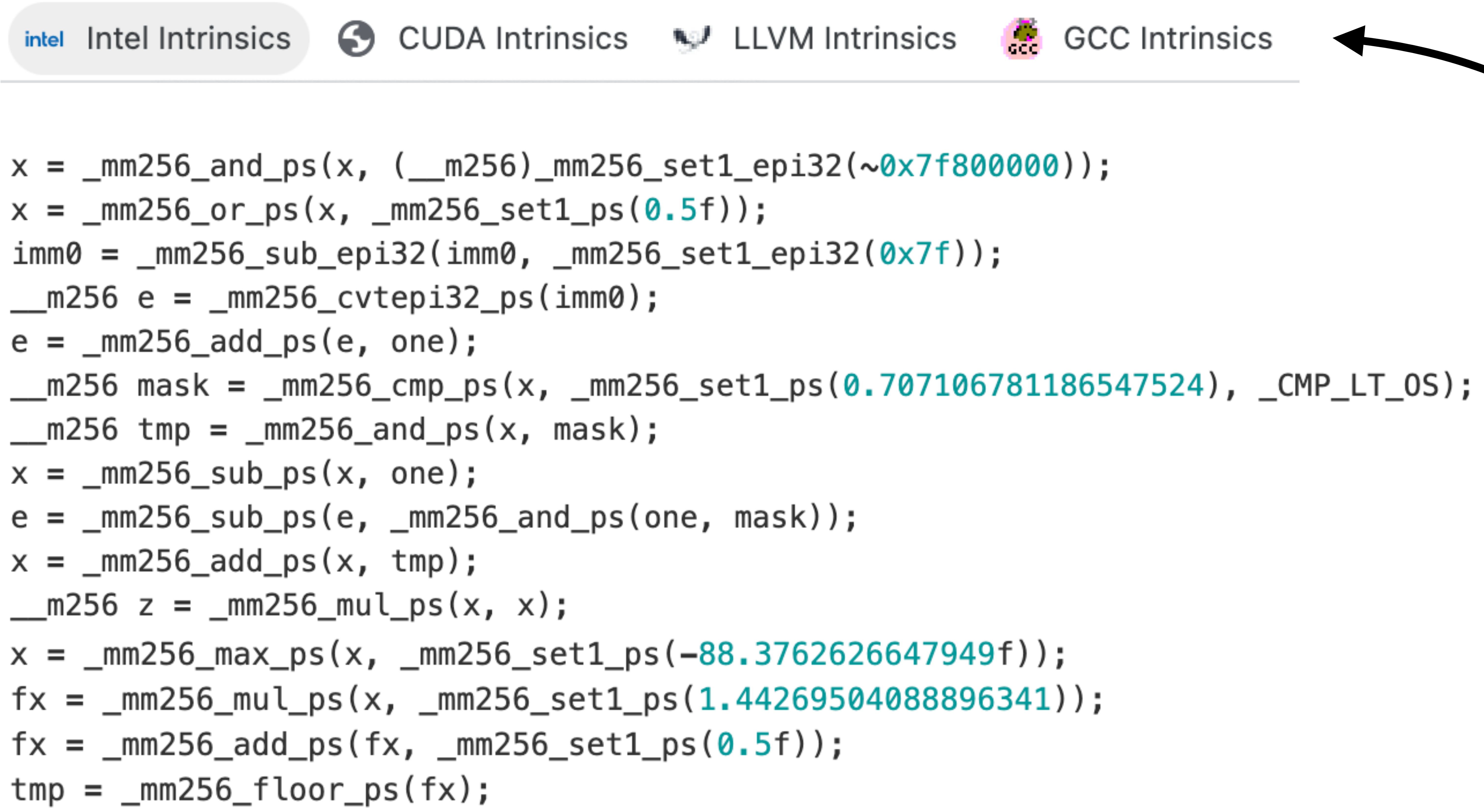
I am Ash

- 2003 - 15
  - Olympiads, Web, iOS, MacOS,
  - Astrophysics & Scientific Computing
- 2015 -
  - Started Unum to build the largest intelligent systems.
  - Worked on Neural Nets, Graphs, Analytics, Compression, Encryption…

@ashvardanian

I was working on nanosecond optimizations

When I faced bottlenecks in storage: Postgres, MongoDB, Neo4J…



```
x = _mm256_and_ps(x, (__m256)_mm256_set1_epi32(~0x7f800000));
x = _mm256_or_ps(x, _mm256_set1_ps(0.5f));
imm0 = _mm256_sub_epi32(imm0, _mm256_set1_epi32(0x7f));
__m256 e = _mm256_cvtepi32_ps(imm0);
e = _mm256_add_ps(e, one);
__m256 mask = _mm256_cmp_ps(x, _mm256_set1_ps(0.707106781186547524), _CMP_LT_OS);
__m256 tmp = _mm256_and_ps(x, mask);
x = _mm256_sub_ps(x, one);
e = _mm256_sub_ps(e, _mm256_and_ps(one, mask));
x = _mm256_add_ps(x, tmp);
__m256 z = _mm256_mul_ps(x, x);
x = _mm256_max_ps(x, _mm256_set1_ps(-88.3762626647949f));
fx = _mm256_mul_ps(x, _mm256_set1_ps(1.44269504088896341));
fx = _mm256_add_ps(fx, _mm256_set1_ps(0.5f));
tmp = _mm256_floor_ps(fx);
```

Browser Homepage

No shortage of alternative databases

Company	Raised in 2021	Total Raised	Valuation	Total Rounds	Raised in 2021, %
CockroachDB	438M	633M	5B	9	69%
Neo4J	392M	582M	3B	10	67%
Clickhouse	300M	300M	2B	2	100%
Yugabyte	188M	291M	2B	5	64%
Redis	111M	356M	5B	10	31%
TigerGraph	105M	171M	1B	6	61%

unum.cloud: DBMS Gold Rush of 2021

Most new Databases grow on Rocks 🪨

LevelDB + Transactions + LSM Tree

- Facebook: MyRocks = MySQL on RocksDB
- Twitter: Manhattan distributed store on RocksDB
- Yahoo: Sherpa distributed store on RocksDB

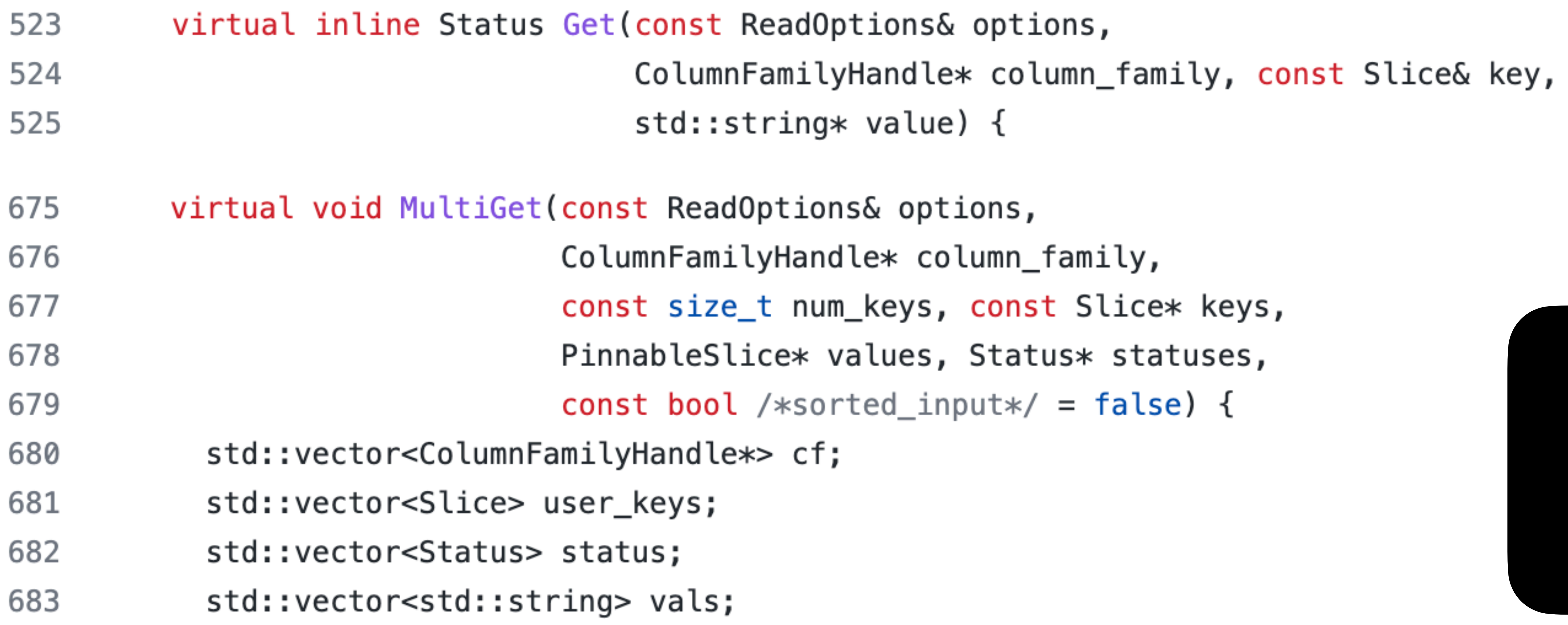
- CockroachDB = Distributed Postgres on RocksDB
- Yugabyte = Distributed Postgres on RocksDB

} Same?

- Apache Samza, Kafka, …

Diving into RocksDB

Felt wrong after SIMD



```
  virtual inline Status Get(const ReadOptions& options,
                            ColumnFamilyHandle* column_family, const Slice& key,
                            std::string* value) {

  virtual void MultiGet(const ReadOptions& options,
                        ColumnFamilyHandle* column_family,
                        const size_t num_keys, const Slice* keys,
                        PinnableSlice* values, Status* statuses,
                        const bool /*sorted_input*/ = false) {
    std::vector<ColumnFamilyHandle*> cf;
    std::vector<Slice> user_keys;
    std::vector<Status> status;
    std::vector<std::string> vals;
```

STL containers ✓

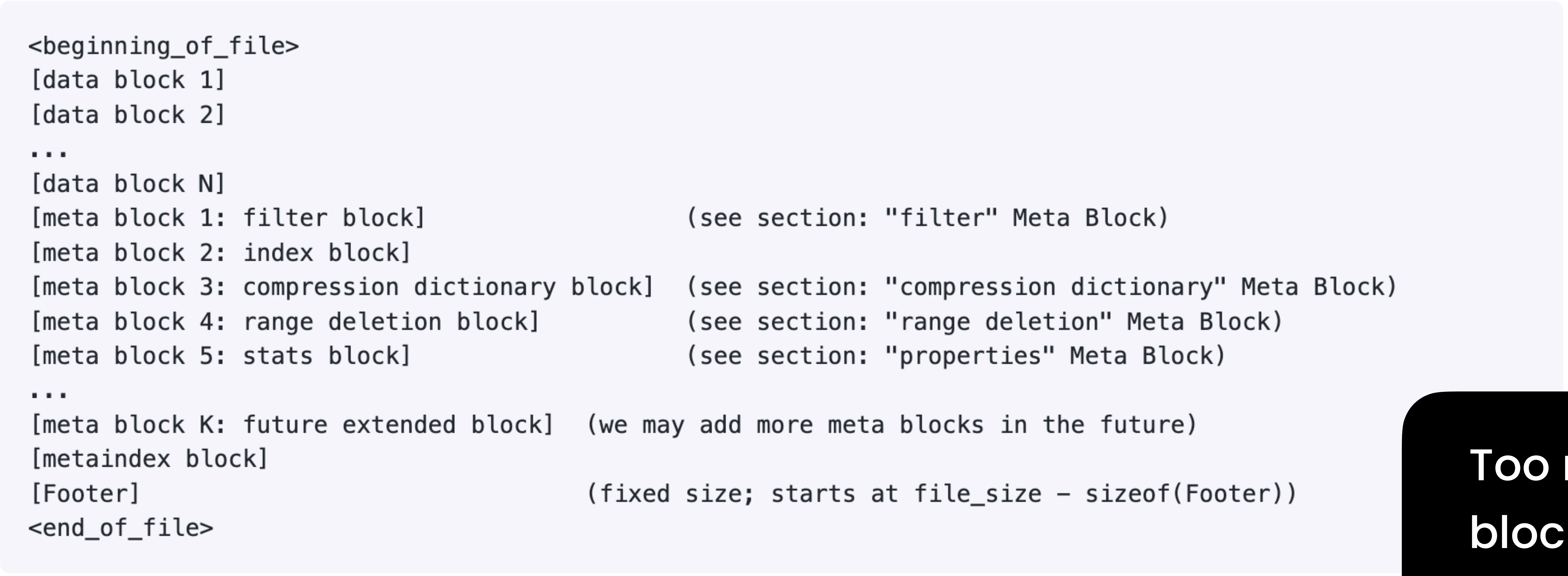
Global allocators ✓

Excessive allocations ✓

rocksdb/include/rocksdb/db.h

Same Story with File Structure

BlockBasedTable Format isn’t NVMe-Friendly



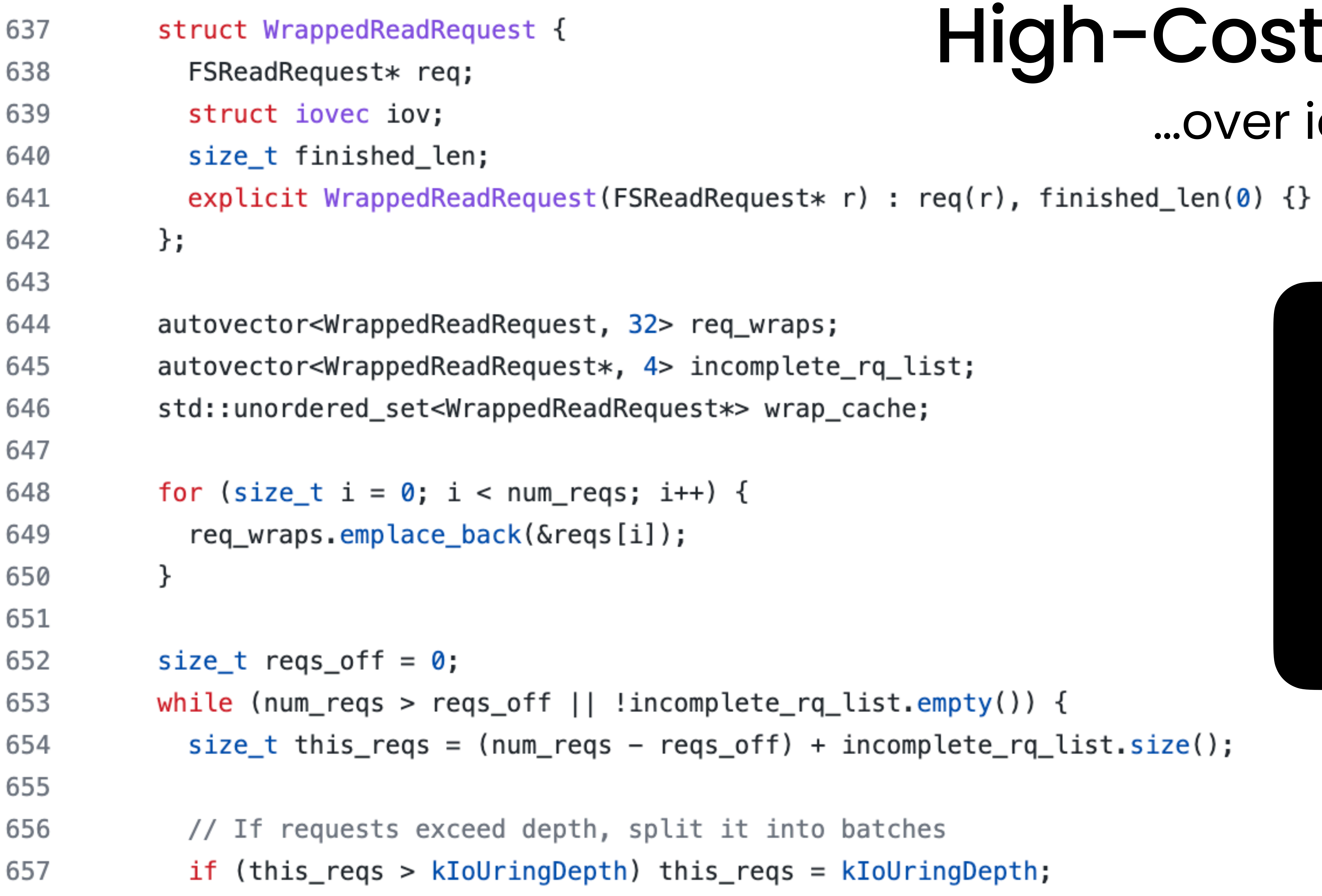
```
<beginning_of_file>
[data block 1]
[data block 2]
...
[data block N]
[meta block 1: filter block]                  (see section: "filter" Meta Block)
[meta block 2: index block]
[meta block 3: compression dictionary block]  (see section: "compression dictionary" Meta Block)
[meta block 4: range deletion block]          (see section: "range deletion" Meta Block)
[meta block 5: stats block]                   (see section: "properties" Meta Block)
...
[meta block K: future extended block]  (we may add more meta blocks in the future)
[metaindex block]
[Footer]                               (fixed size; starts at file_size - sizeof(Footer))
<end_of_file>
```

Too many functional blocks compensating for poor design choices

rocksdb/wiki/Rocksdb-BlockBasedTable-Format

High-Cost Abstractions

…over io_uring and liburing

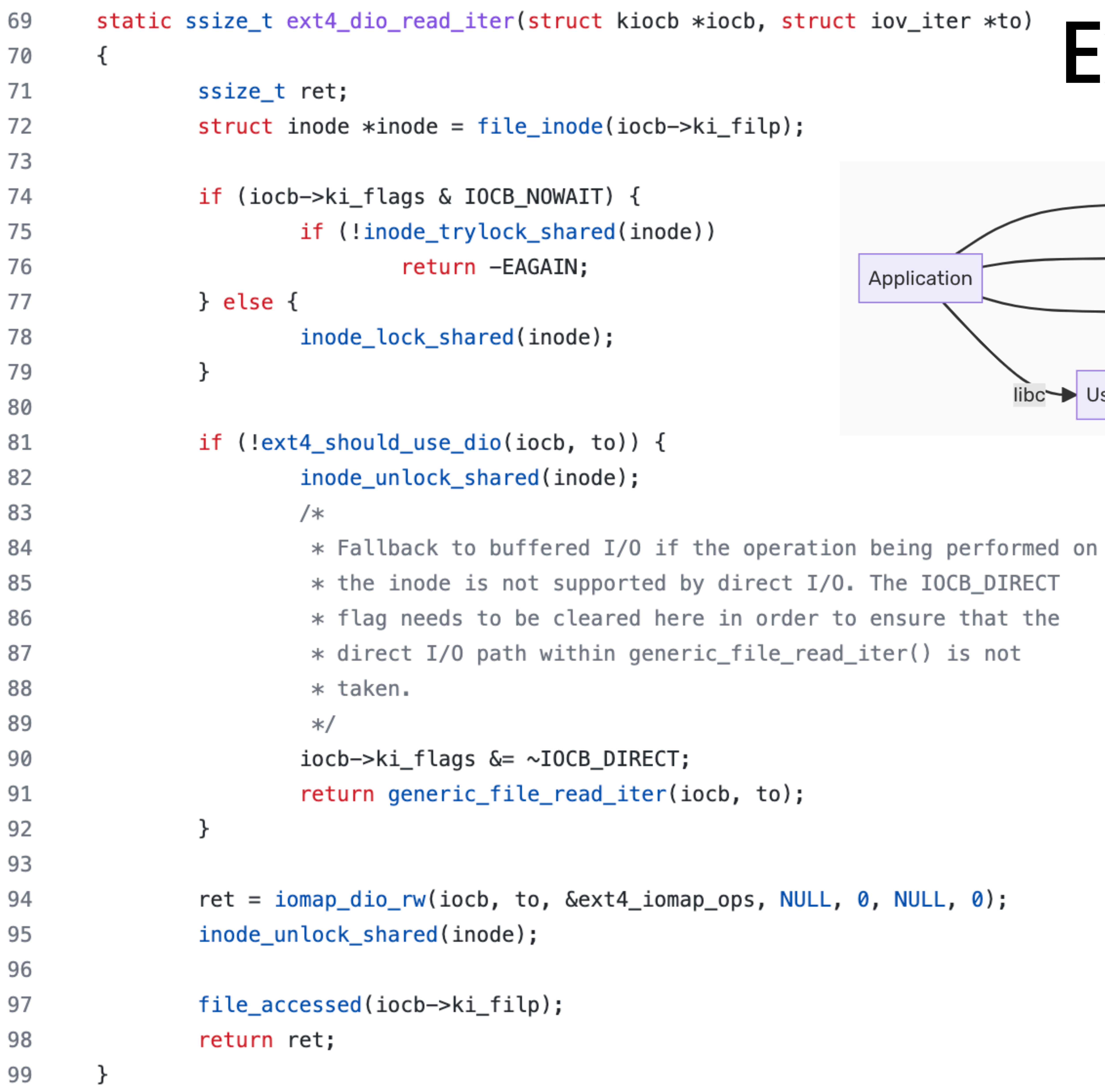


```
struct WrappedReadRequest {
  FSReadRequest* req;
  struct iovec iov;
  size_t finished_len;
  explicit WrappedReadRequest(FSReadRequest* r) : req(r), finished_len(0) {}
};

autovector<WrappedReadRequest, 32> req_wraps;
autovector<WrappedReadRequest*, 4> incomplete_rq_list;
std::unordered_set<WrappedReadRequest*> wrap_cache;

for (size_t i = 0; i < num_reqs; i++) {
  req_wraps.emplace_back(&reqs[i]);
}

size_t reqs_off = 0;
while (num_reqs > reqs_off || !incomplete_rq_list.empty()) {
  size_t this_reqs = (num_reqs - reqs_off) + incomplete_rq_list.size();

  // If requests exceed depth, split it into batches
  if (this_reqs > kIoUringDepth) this_reqs = kIoUringDepth;
```

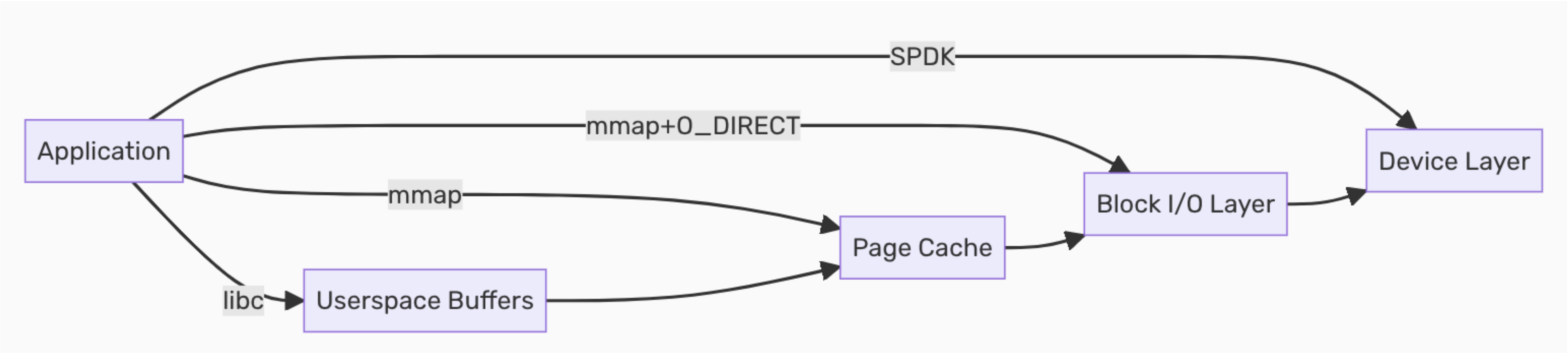
Wrapping requests with metadata negates the benefits of deep queues with heap-allocated vectors and complex sync logic

rocksdb/env/io_posix.cc

Ext4 Filesystem Example



```
static ssize_t ext4_dio_read_iter(struct kiocb *iocb, struct iov_iter *to)
{
        ssize_t ret;
        struct inode *inode = file_inode(iocb->ki_filp);

        if (iocb->ki_flags & IOCB_NOWAIT) {
                if (!inode_trylock_shared(inode))
                        return -EAGAIN;
        } else {
                inode_lock_shared(inode);
        }

        if (!ext4_should_use_dio(iocb, to)) {
                inode_unlock_shared(inode);
                /*
                 * Fallback to buffered I/O if the operation being performed on
                 * the inode is not supported by direct I/O. The IOCB_DIRECT
                 * flag needs to be cleared here in order to ensure that the
                 * direct I/O path within generic_file_read_iter() is not
                 * taken.
                 */
                iocb->ki_flags &= ~IOCB_DIRECT;
                return generic_file_read_iter(iocb, to);
        }

        ret = iomap_dio_rw(iocb, to, &ext4_iomap_ops, NULL, 0, NULL, 0);
        inode_unlock_shared(inode);

        file_accessed(iocb->ki_filp);
        return ret;
}
```



Most modern IO goes through more layers, than presented on diagram, locking mutexes everywhere.

linux/fs/ext4/file.c

Modern Key-Value Stores at Glance

Have three parts

- **Concurrent Mem-Table**: allocator-dependent Skip-List
- **Versioning & Garbage Collection**: slow compactions
- **IO Logic**: synchronous, interrupting, or poor async

Topic of Today

Which are the IO options?

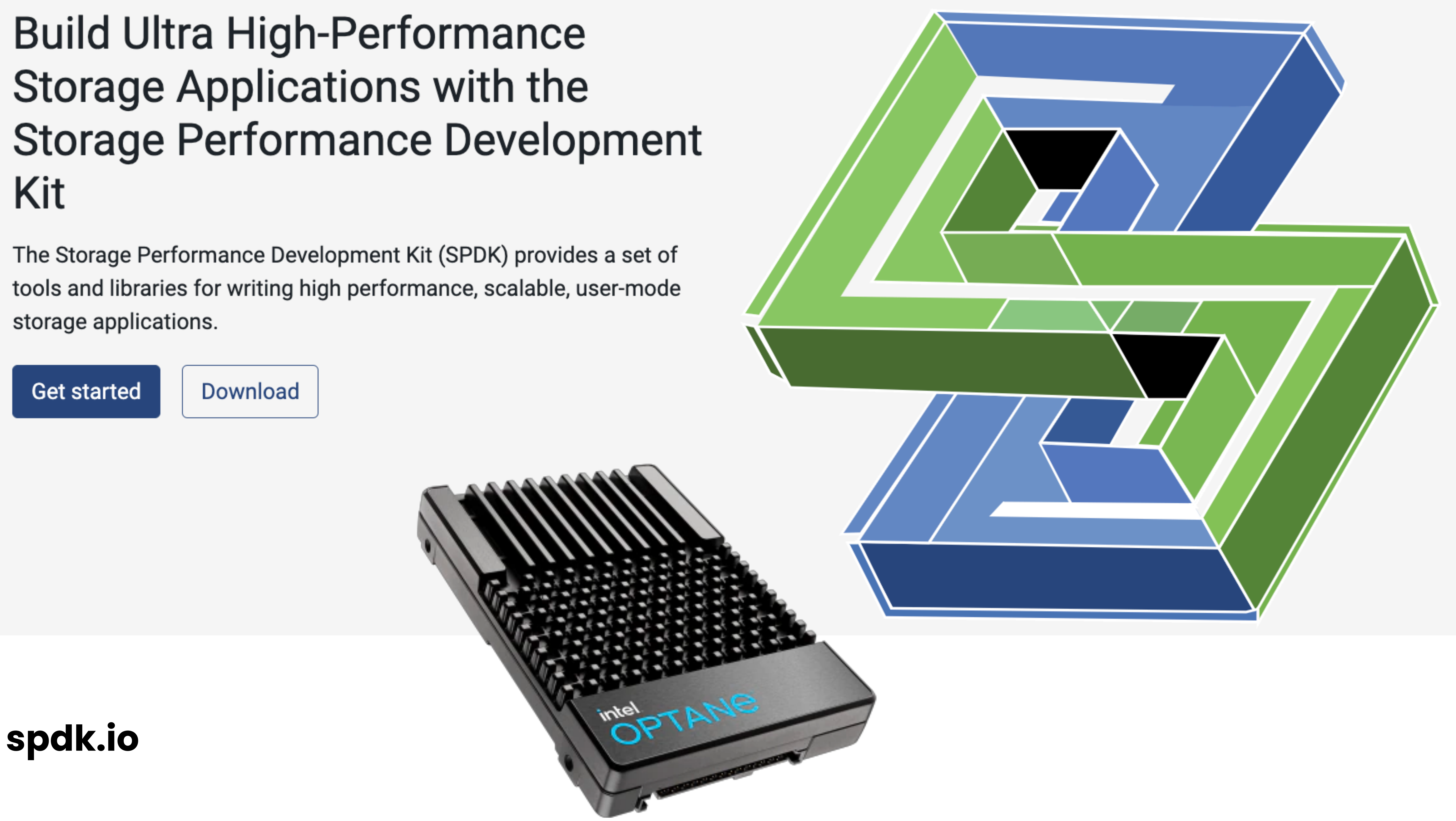
In order of maturity

- **UNIX IO** system calls
- **POSIX AIO** since Linux kernel 2.5 ~ 2002
- **io_uring** since Linux kernel 5.1 ~ 2019
- **Magnum IO** for Nvidia GPUs, including GPU Direct Storage
- **SPDK** on Linux

Sub-Topic of Today

Intel & Micron announced 3D XPoint in 2015

But they had no IO stack ready for 5 µs devices



Build Ultra High-Performance Storage Applications with the Storage Performance Development Kit

The Storage Performance Development Kit (SPDK) provides a set of tools and libraries for writing high performance, scalable, user-mode storage applications.

spdk.io

SPDK Hello World

6 steps, 500 lines of code 😅

1. 🆕 **Root** privileges
2. 🆕 **Probe** for NVMe controllers
3. 🆕 Create multiple **non-thread-safe** IO queues per controller
4. 🆕 Allocate page-aligned buffers with **pinned** addresses
5. Submit requests
6. Poll for completion

spdk/examples/nvme/hello_world/hello_world.c

To squeeze everything from SPDK

You should:

- Forget about **filesystem**

SPDK gives you a raw block device.

You don’t have filenames, nested paths, etc.

But you also don’t pay for tons of legacy synchronous FS code.

To squeeze everything from SPDK

You should:

- Forget about filesystem
- Forget about page-**caching**

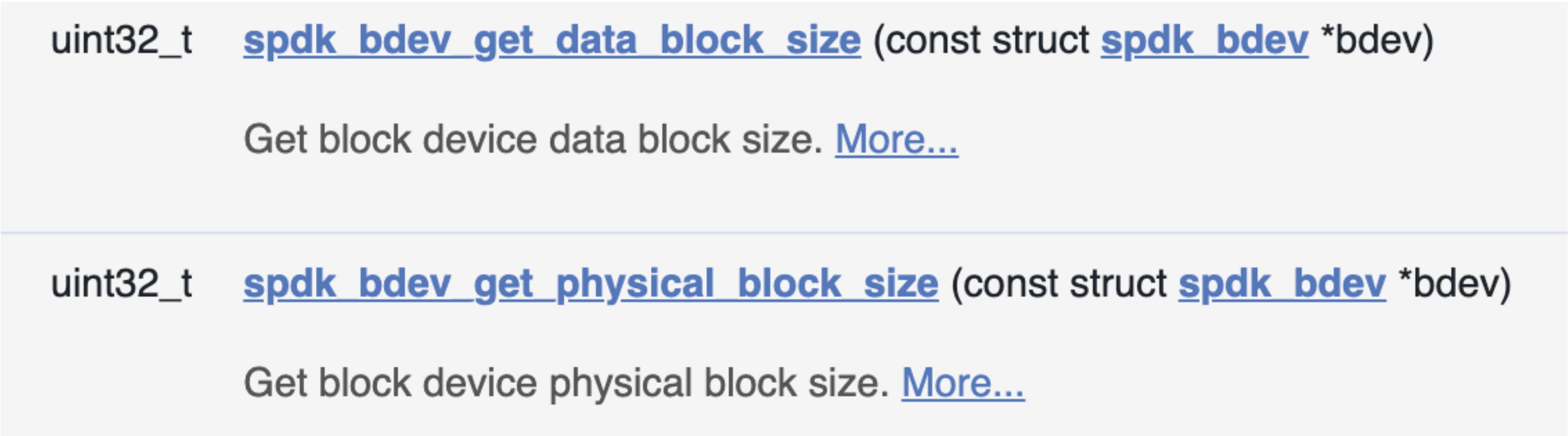
Everything is designed for `O_DIRECT`, so you don’t pay for `kswapd0`.

Need a cache - write one.

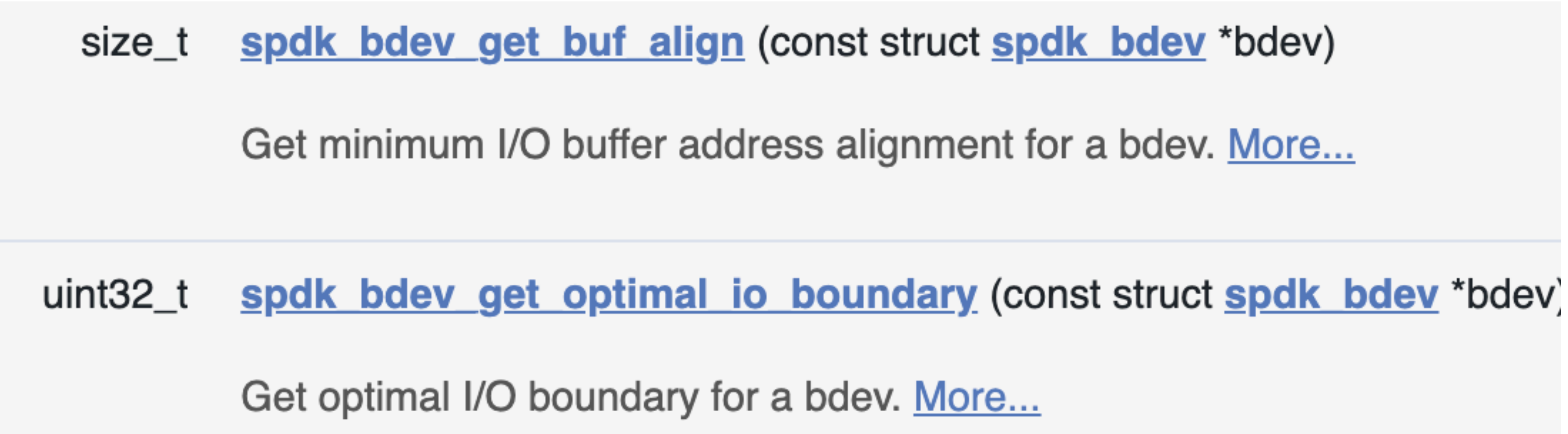
To squeeze everything from SPDK

You should:

- Forget about filesystem
- Forget about page-caching
- Forget about addressing **bytes**, and focus on **pages**



```
uint32_t   spdk_bdev_get_data_block_size (const struct spdk_bdev *bdev)

           Get block device data block size. More...

uint32_t   spdk_bdev_get_physical_block_size (const struct spdk_bdev *bdev)

           Get block device physical block size. More...
```



```
  size_t   spdk_bdev_get_buf_align (const struct spdk_bdev *bdev)

           Get minimum I/O buffer address alignment for a bdev. More...

uint32_t   spdk_bdev_get_optimal_io_boundary (const struct spdk_bdev *bdev)

           Get optimal I/O boundary for a bdev. More...
```

Let’s benchmark

On bare metal, no RAID

AMD Threadripper PRO 3995WX

128 threads @ 2.7 GHz

8x Samsung M393AAG40M32-CAE

1 TB RAM @ 3.2 GHz, 204 GB/s

8x Samsung PM1733 U.2

64 TB NVMe @ 48 GB/s

4x Nvidia RTX 3090

Let’s benchmark

On bare metal, no RAID

- UNIX IO: **50.7k** IOPS
  - On 1 SSD

Let’s benchmark

On bare metal, no RAID

- UNIX IO: 50.7k IOPS
- POSIX AIO: **573k** IOPS
  - On 1 SSD

Let’s benchmark

On bare metal, no RAID

- UNIX IO: 50.7k IOPS
- POSIX AIO: 573k IOPS
- io_uring: **869k** IOPS
    - Over **5M** IOPS on 8 SSDs with 24 threads

Let’s benchmark

On bare metal, no RAID

- UNIX IO: 50.7k IOPS
- POSIX AIO: 573k IOPS
- io_uring: 869k IOPS
- SPDK: **1.2M** IOPS
  - Over **9M** IOPS on 8 SSDs with 24 threads

No native SPDK support in `fio`, only through xNVME.

Real World Performance

From Synthetic IO to KVS and DBMS

Engine	Random Batch Writes	Random Batch Reads
RocksDB	57,000 ~ **200 MB/s**	650,000 ~ **2.6 GB/s**
UDisk	320,000 ~ **1.3 GB/s ~ 5.8x**	4,200,000 ~ **16.8 GB/s ~ 6.5x**

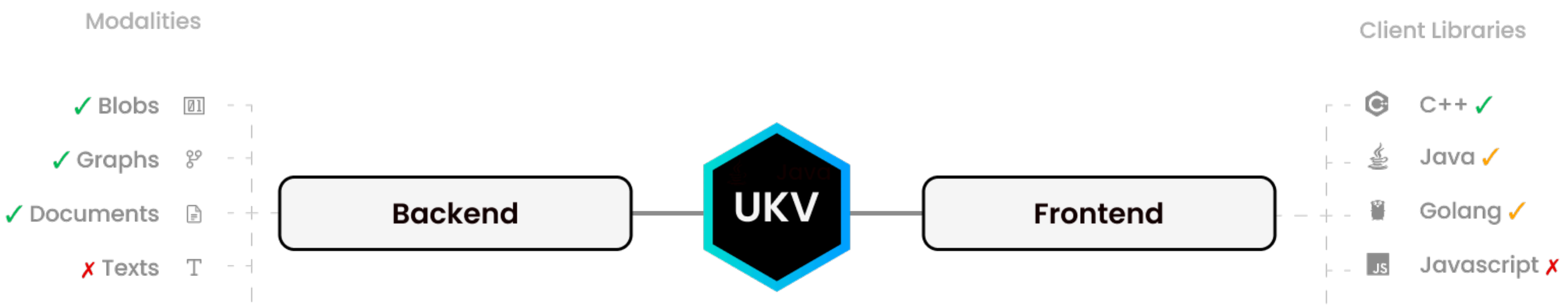
On 10 TB collections, with 1 TB of RAM, 8x SSDs and 32 cores

With 8 byte keys and misaligned direct accesses

unum.cloud/ucsb

UKV: The BLAS of CRUD

Open Binary Interface Standard



Modalities

✓ Blobs

✓ Graphs

✓ Documents

✗ Texts

✓ Vectors

Backend

UKV

Frontend

Client Libraries

C++ ✓

Java ✓

Golang ✓

Javascript ✗

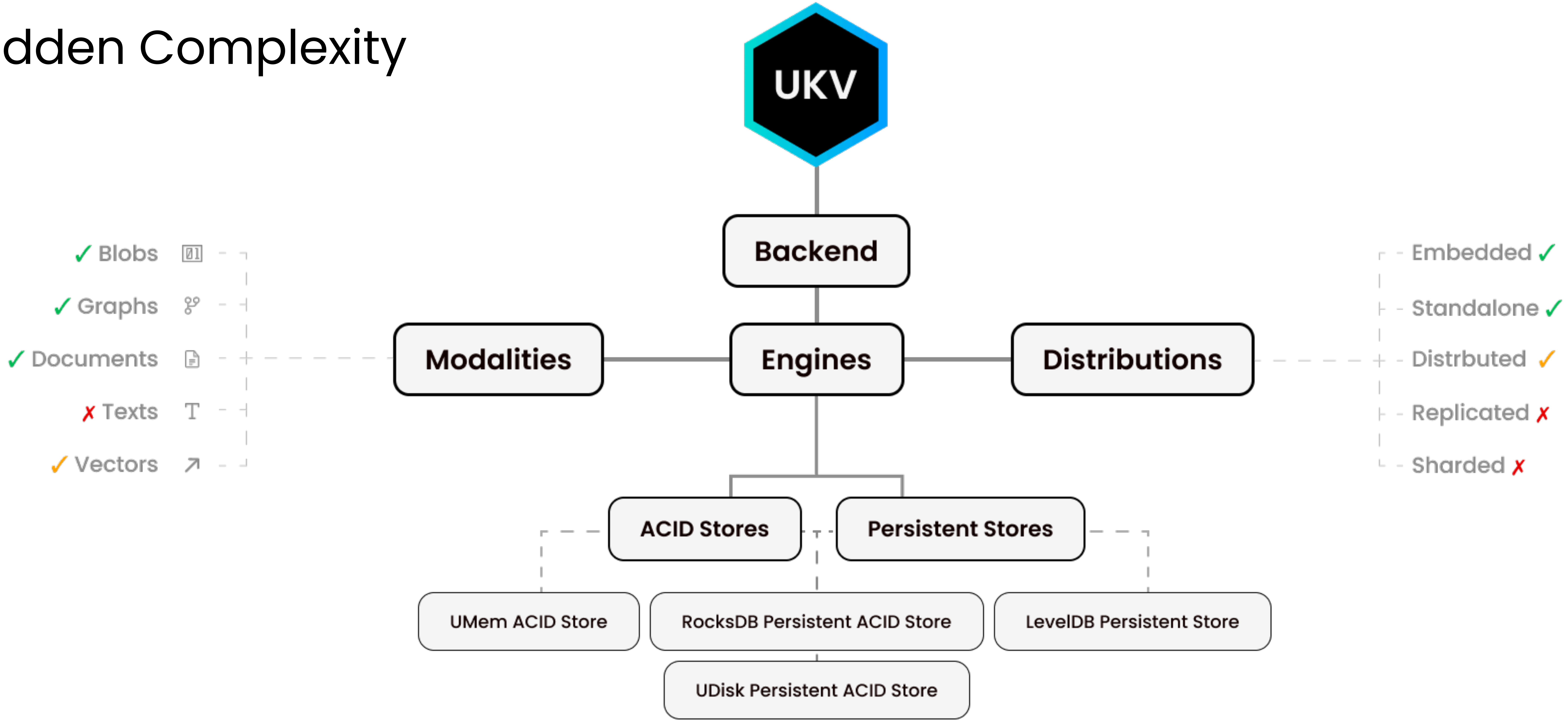
Python ✓

Started in Summer 2022

github.com/unum-cloud/ukv

UKV Backends

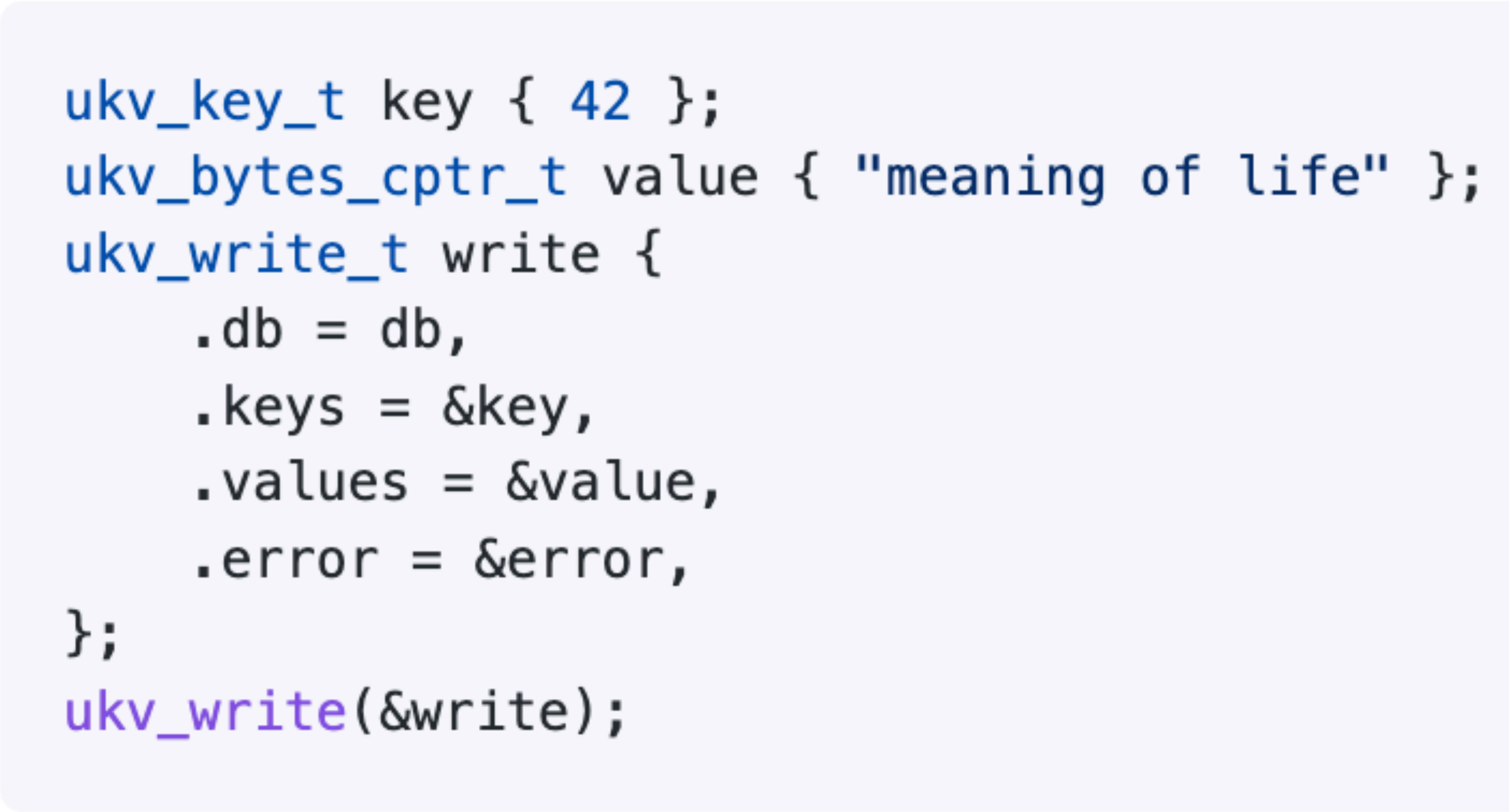
Hidden Complexity



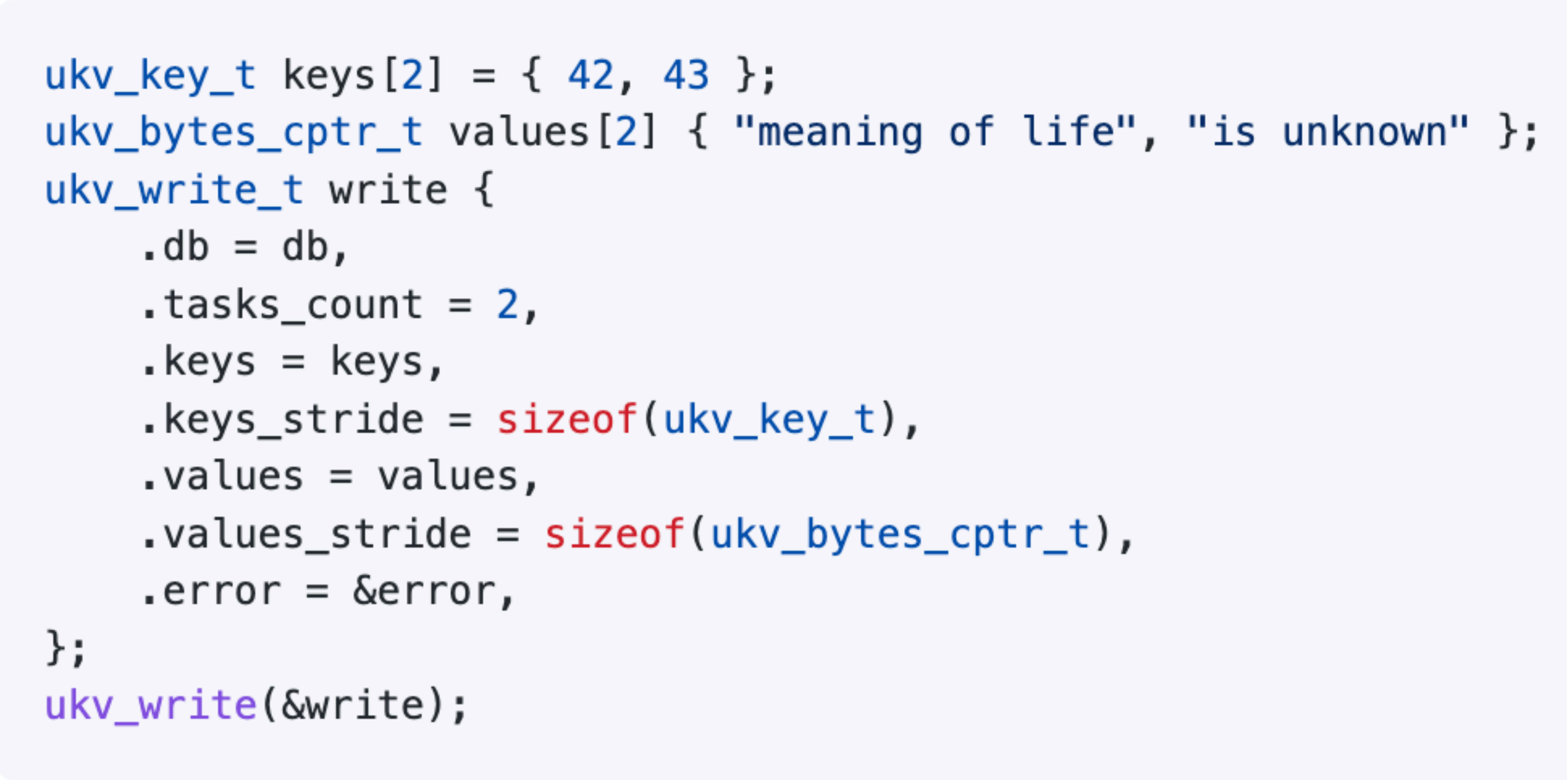
github.com/unum-cloud/ukv

UKV C Standard

Supports **strides**, like BLAS



```c
ukv_key_t key { 42 };
ukv_bytes_cptr_t value { "meaning of life" };
ukv_write_t write {
    .db = db,
    .keys = &key,
    .values = &value,
    .error = &error,
};
ukv_write(&write);
```

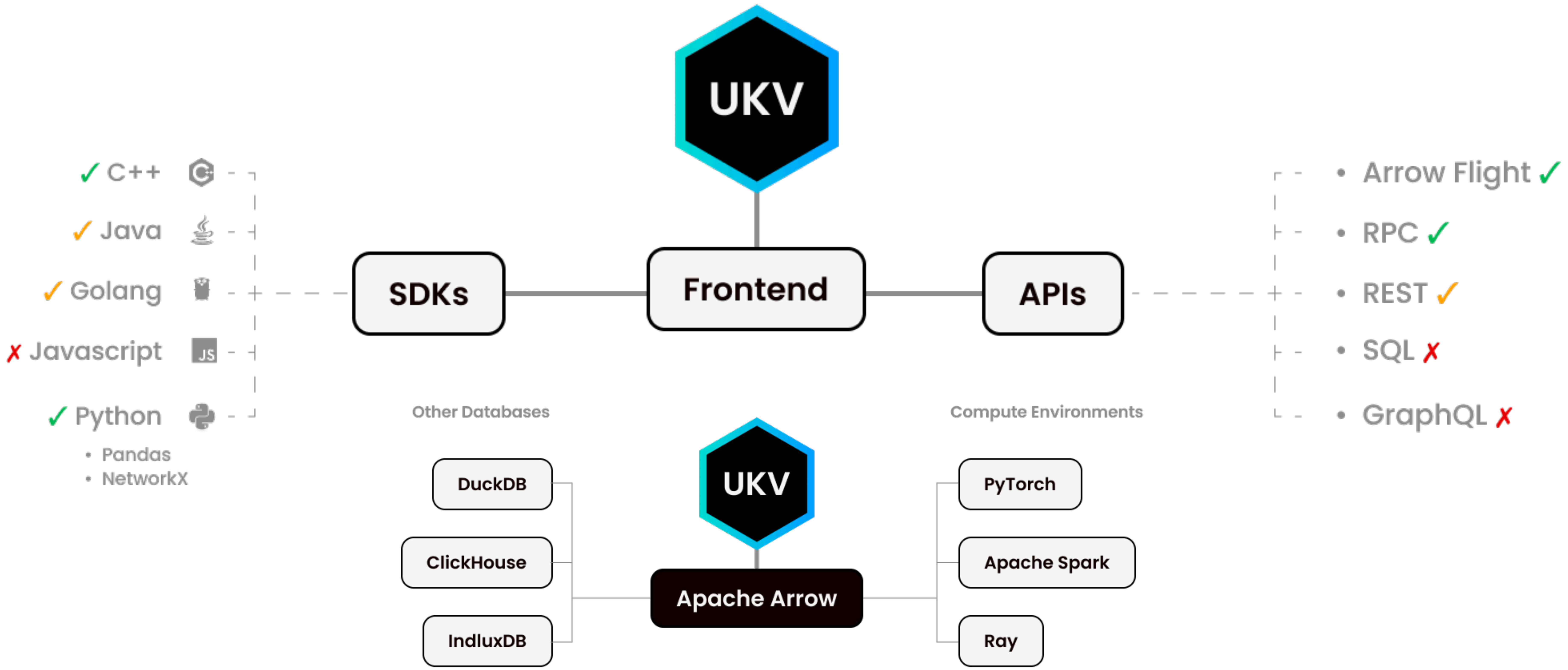


```c
ukv_key_t keys[2] = { 42, 43 };
ukv_bytes_cptr_t values[2] { "meaning of life", "is unknown" };
ukv_write_t write {
    .db = db,
    .tasks_count = 2,
    .keys = keys,
    .keys_stride = sizeof(ukv_key_t),
    .values = values,
    .values_stride = sizeof(ukv_bytes_cptr_t),
    .error = &error,
};
ukv_write(&write);
```

github.com/unum-cloud/ukv/include/ukv/blobs.h

UKV Frontends

Performance is Accessible



UKV

Frontend

SDKs

- ✓ C++
- ✓ Java
- ✓ Golang
- ✗ Javascript
- ✓ Python
  - Pandas
  - NetworkX

APIs

- Arrow Flight ✓
- RPC ✓
- REST ✓
- SQL ✗
- GraphQL ✗

Other Databases

- DuckDB
- ClickHouse
- IndluxDB

UKV

Apache Arrow

Compute Environments

- PyTorch
- Apache Spark
- Ray

github.com/unum-cloud/ukv

UKV Python SDK

Performance is Accessible

```
main_collection[42] = binary_string
main_collection.set(42, binary_string)
```

```
42 in main_collection
main_collection.has_key(42)
```

```
main_collection[42]
main_collection.get(42)
```

```
del main_collection[42]
main_collection.pop(42)
```

```
main_collection[[42, 43, 44]]
main_collection[(42, 43, 44)]
```

ARROW

```
import pyarrow as pa
keys = pa.array([1000, 2000], type=pa.int64())
strings: pa.StringArray = pa.array(['some', 'text'])
main_collection[keys] = strings
```

```
rows_batch = main_collection.sample(1_000)
values_batch = main_collection.docs.table[['name', 'age']].loc[rows_batch]
```

RAPIDS

pandas

NetworkX
Network Analysis in Python

github.com/unum-cloud/ukv

Give it a try

And join the development!

unum-cloud/ukv

pip install ukv

t.me/cpparm

Linux, GCC, C++, Python: Today

MSVC, AppleClang, GoLang, Java: Soon

@ashvardanian

Check out Unum.Cloud

GitHub.com/Unum-Cloud/UKV

@ashvardanian

++ High Load Armenia

Co-organizer

Yandex